Complexity Theory

Lecture 10: Time vs Space Complexity

Tom Gur

http://www.cl.cam.ac.uk/teaching/2324/Complexity

Sublinear Complexity!

Space Complexity

We've already seen the definition $\mathrm{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

Space Complexity

We've already seen the definition $\mathrm{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

Counting only work space.

Space Complexity

We've already seen the definition $\mathrm{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

Counting only work space.

$\mathrm{NSPACE}(f)$ is the class of languages accepted by a *nondeterministic* Turing machine using at most $O(f(n))$ work space.

Space Complexity

We've already seen the definition $\mathrm{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

Counting only work space.

$\mathrm{NSPACE}(f)$ is the class of languages accepted by a *nondeterministic* Turing machine using at most $O(f(n))$ work space.

As we are only counting work space, it makes sense to consider bounding functions f that are less than linear.

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

co-NL – the languages whose complements are in NL.

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

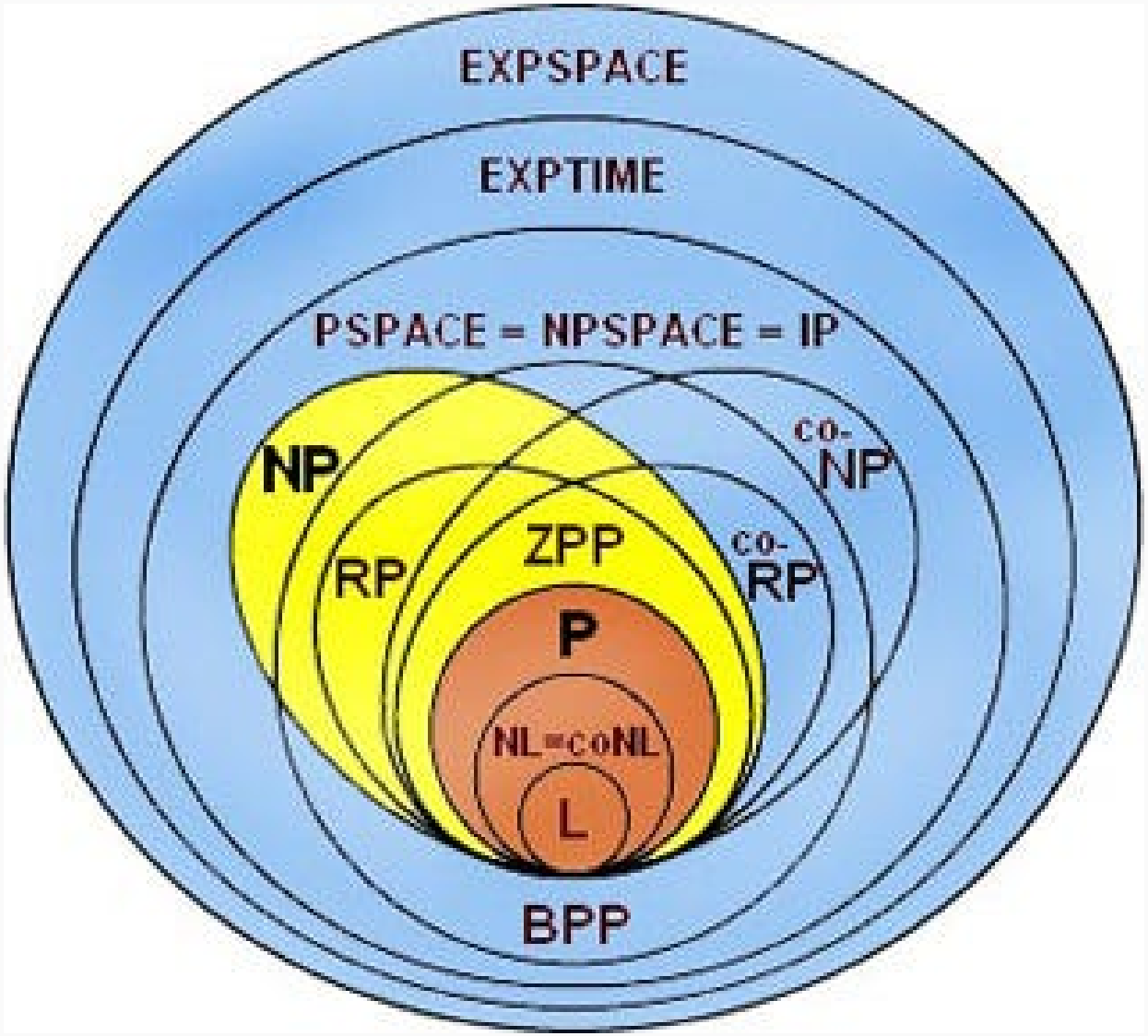
$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

co-NL – the languages whose complements are in NL.

co-NPSPACE – the languages whose complements are in NPSPACE.

Space Complexity Zoo



Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

and $\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

and $\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \text{co-NL}$$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

and $\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \text{co-NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \text{co-NP}$$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

and $\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \text{co-NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \text{co-NP}$$

$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \cap \text{co-NPSPACE}$$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

and $\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \text{co-NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \text{co-NP}$$

$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \cap \text{co-NPSPACE}$$

It would be easier to prove a more general statement!

Constructible Functions

To prove more general inclusion, we restrict our attention to reasonable time functions.

Constructible Functions

To prove more general inclusion, we restrict our attention to reasonable time functions.

A complexity class such as $\mathrm{TIME}(f)$ can be very unnatural, if f is.

Constructible Functions

To prove more general inclusion, we restrict our attention to reasonable time functions.

A complexity class such as $\mathrm{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Constructible Functions

To prove more general inclusion, we restrict our attention to reasonable time functions.

A complexity class such as $\mathrm{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and

Constructible Functions

To prove more general inclusion, we restrict our attention to reasonable time functions.

A complexity class such as $\mathrm{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and
- there is a deterministic machine M which, on any input of length n, replaces the input with the string $0^{f(n)}$, and M runs in time $O(n + f(n))$ and uses $O(f(n))$ *work space*.

Examples

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

If f and g are constructible functions, then so are $f + g$, $f \cdot g$, 2^f and $f(g)$ (this last, provided that $f(n) > n$).

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every x, there is an accepting computation of M on x of length at most $O(f(n))$.

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every x, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in NTIME(f) is accepted by a machine for which all computations are of length at most $O(f(n))$.

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every x, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in NTIME(f) is accepted by a machine for which all computations are of length at most $O(f(n))$.

Also, given a Turing machine M and a constructible function f, we can define a machine that simulates M for $f(n)$ steps.

‘

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

The first two are straightforward from definitions.

The third is an easy simulation.

The last requires some more work.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.

Algorithm?

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t. Algorithm?

A simple search algorithm solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t. Algorithm?

A simple search algorithm solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t. Algorithm?

A simple search algorithm solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t. Algorithm?

A simple search algorithm solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

Complexity?

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

Complexity?

Bonus: Can you do it in NL?

Nondeterministic space vs deterministic time

We can use the $O(n^2)$ time algorithm for st-Connectivity to show that:

$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$

for some constant k.

Nondeterministic space vs deterministic time

We can use the $O(n^2)$ time algorithm for st-Connectivity to show that:

$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$

for some constant k.

Let M be a nondeterministic machine working in space bounds $f(n)$.

Nondeterministic space vs deterministic time

We can use the $O(n^2)$ time algorithm for st-Connectivity to show that:

$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$

for some constant k.

Let M be a nondeterministic machine working in space bounds $f(n)$.

For any input x of length n, there is a constant c (depending on the number of states and alphabet of M) such that the total number of possible configurations of M within space bounds $f(n)$ is bounded by $n \cdot c^{f(n)}$.

Nondeterministic space vs deterministic time

We can use the $O(n^2)$ time algorithm for st-Connectivity to show that:

$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$

for some constant k.

Let M be a nondeterministic machine working in space bounds $f(n)$.

For any input x of length n, there is a constant c (depending on the number of states and alphabet of M) such that the total number of possible configurations of M within space bounds $f(n)$ is bounded by $n \cdot c^{f(n)}$.

> *Here, $c^{f(n)}$ represents the number of different possible contents of the work space, and n different head positions on the input.*

Configuration Graph

Define the *configuration graph* of M, x to be the graph whose nodes are the possible configurations, and there is an edge from i to j if, and only if, $i \to_M j$.

Configuration Graph

Define the *configuration graph* of M, x to be the graph whose nodes are the possible configurations, and there is an edge from i to j if, and only if, $i \to_M j$.

Then, M accepts x if, and only if, some accepting configuration is reachable from the starting configuration $(s, \triangleright, x, \triangleright, \varepsilon)$ in the configuration graph of M, x.

Space vs Time

Using the $O(n^2)$ algorithm for st-Connectivity, we get that $L(M)$—the language accepted by M—can be decided by a deterministic machine operating in time

Space vs Time

Using the $O(n^2)$ algorithm for st-Connectivity, we get that $L(M)$—the language accepted by M—can be decided by a deterministic machine operating in time

$$O(|G|^2) = O((nc^{f(n)})^2) = O(k^{(\log n + f(n))})$$

Space vs Time

Using the $O(n^2)$ algorithm for st-Connectivity, we get that $L(M)$—the language accepted by M—can be decided by a deterministic machine operating in time

$$O(|G|^2) = O((nc^{f(n)})^2) = O(k^{(\log n + f(n))})$$

In particular, this establishes that NL $\subseteq$ P and NPSPACE $\subseteq$ EXP.

Scaling up complexity results

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Thus $S \in \mathsf{PSPACE}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Thus $S \in \mathsf{PSPACE}$.

A similar argument shows that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{EXP} = \mathsf{NEXP}$.

Bonus: Interactive Proofs (IP) and PCPs